An Introduction to Trilinos

Michael A. Heroux

Solvers Project Leader

Sandia National Labs

ACTS Workshop October 11, 2001

Outline

- Preliminary mention of “class classifications.”
- Overview of Trilinos components.
- Using Epetra/AztecOO for solving linear systems.
- Epetra base and utility classes.
- Briefly: Epetra Import/Export Capabilities.
- Introduction to TSF Abstract Class Hierarchy.
- Status and availability of Trilinos components.

Class Classifications: Interface vs. Implementation

- Modern languages (C++, Java, F95, C#?) provide powerful capabilities to separate:
  - Interface: What should be done with whom.
  - Implementation: How it is done.
- Concept is not new. Example: BLAS.
  - Single Fortran interface.
    - Reference Fortran implementation.
    - Numerous assembly language implementations.
    - C implementation in ATLAS.
  - However, Non-Fortran implementations not portable.

Sandia National Laboratories

Class Classifications

- **Abstract class: Interface only (just header files).**
  - Cannot construct them.
  - Can be passed in as arguments.
  - Can call methods from this class.
  - Methods tend to be generic, conceptual in nature.
  - Bottom line: User focused, attention to the big picture.
- **Concrete class: Interface and code.**
  - Can be constructed, passed in, methods used.
  - Provide specific functionality and interfaces, powerful capabilities.
  - Bottom line: Internally focused, attention to the detailed picture.
- **Adaptor class: Glue to combine abstract and concrete.**
  - Combines the best of both class types:
    - Abstract class interface.
    - Capabilities of concrete class.
  - Hides the implementation details from abstract class user.
  - Support multiple concrete implementations via multiple adaptors.

Sandia National Laboratories

Trilinos Classes

- Trilinos provides all three types of classes:
  - **Abstract:**
    - Equation Solver Interface (ESI)
      - *de facto* standard for abstract linear solver interfaces.
    - Finite Element Interface (FEI)
      - Commonly used interface for finite element codes.
    - Trilinos Solver Framework (TSF)
      - Extensive, multi-purpose abstract class hierarchy.
  - **Concrete:**
    - Many, listed next…
  - **Adaptor:**
    - Many.
    - Implement each class of ESI, FEI and TSF using one or more appropriate concrete classes.
    - Implement TSF using third party libraries, e.g., PETSc, SuperLU, etc.

What is Trilinos?

- Trilinos[1] is Sandia's multifaceted solver project.
- Encompasses efforts in:
  - Linear solvers.
  - Eigen solvers.
  - Nonlinear and time-dependent solvers.
  - Others.

- Provides a common framework for current and future solver projects at Sandia.

- Specifically provides:
  - A common set of concrete linear algebra objects for solver development and application interfaces.
  - A consistent set of solver interfaces via abstract classes (API) .

[1]Trilinos, pronounced tree-lee-nose, is a Greek word that, loosely translated, means a "string of pearls".

Sandia National Laboratories

The Trilinos Solver Framework (TSF)

- TSF specifies a set of abstract classes for:
  - ParameterList (simple database).
  - Multivector (generalization of vector).
  - Operator. (base transformation class).
  - Problem (primary control class).

  - And specializations of these classes.

- These interfaces prescribe:
  - What these objects should do.
  - How they are related.

- *But do not specify the implementation.*

Sandia National Laboratories

Trilinos Concrete Solver Components

- **Linear systems:**
  - Multi-level preconditioners (ML: Tuminaro, Hu, Howle).
  - Robust algebraic preconditioners (IFPACK: Heroux).
  - Complex solvers (Komplex: Heroux, Day).
  - Block iterative methods (BGMRES, BLCG: Barth, Lehoucq, Heroux).
  - Object-oriented C++ AZTEC (AztecOO: Heroux).
- **Eigen systems:**
  - Scalable generalized eigensolver (ANASAZI: Lehoucq).
- **Nonlinear systems:**
  - Suite of nonlinear methods (NLS: Pawlowski, Kolda, Shadid).

Trilinos Concrete Support Component: Petra

Petra[1] provides distributed matrix and vector services:

- **Construction of and operations with matrices, vectors and graphs.**
- **Parallel redistribution of all these objects (including a Zoltan interface).**
- **All Trilinos solver components understand and use Petra matrices and vectors.**

Three version under development:

- **Epetra (*Essential Petra*):**
  - **Under development for the past 18 months.**
  - **Restricted to real, double precision arithmetic.**
  - **Uses stable core subset of C++.**
  - **Interfaces accessible to C and Fortran users.**
- **Tpetra (*Templated Petra*):**
  - **Next generation C++ version.**
  - **Templated scalar fields (and perhaps ordinal fields).**
  - **Uses namespaces, and STL: Improved usability/efficiency.**
- **Jpetra (*Java Petra*):**
  - **Pure Java. Completely portable to any JVM.**
  - **Interfaces with Java versions of MPI, LAPACK and BLAS.**

[1]Petra is Greek for "foundation".

petra

Sandia National Laboratories

Epetra/AztecOO

- Much of the remaining talk focuses on Epetra/AztecOO.
- Reasons:
  - These two components will be released shortly.
  - These components are closely related to Aztec:
    - Should be of interest to current Aztec users.
  - Investment in Epetra is:
    - Primary hurdle to using any Trilinos component.
    - Easily leveraged to use other Trilinos components as they are released.
    - Possibly useful independent of the rest of Trilinos.

Solving Linear Systems via Epetra/AztecOO

- **Goal:**

Solve $Ax = b$, using Epetra/AztecOO.

- **Proceed step-by-step through the following classes:**
  - **Comm: Defines parallel machine.**
  - **Map: Defines data distribution.**
  - **Vector: Defines RHS/LHS vectors.**
  - **Matrix: Defines Linear Operator**
  - **Problem: Combines pieces to define linear problem.**
  - **AztecOO: Solves linear problem.**

Epetra Details

- Epetra contains constructors and utility routines for:
  - Distributed dense multivectors and vectors.
  - Local replicated multivectors, vectors.
  - Distributed Sparse Graphs and Matrices.
- Written in C++.
- C/Fortran wrapper functions provide access to library.

Epetra User Class Categories

- **Parallel Machine:** Comm, SerialComm, MpiComm, MpiSmpComm
- **Data Layout:** Map, BlockMap, LocalMap
- **Vectors:** Vector, MultiVector
- **Graphs:** CrsGraph
- **Sparse Matrices:** RowMatrix, CrsMatrix, VbrMatrix
- **Aggregates:** LinearProblem
- **Utilities:** Time, Flops
- **Redistribution:** Import, Export, LbGraph, LbMatrix
- **Dense Matrices:** DenseMatrix, DenseVector, BLAS, LAPACK, SimpleSerialDenseSolver, HardSerialDenseSolver
- **Solver:** AztecOO (not part of Epetra, but related).

Sandia National Laboratories

Epetra Communication Classes

- Epetra_Comm is a pure virtual class:
  - Has no executable code: Interfaces only.
  - Encapsulates behavior and attributes of the parallel machine.
  - Defines interfaces for basic services such as:
    - Collective communications.
    - Gather/scatter capabilities.
  - Allows multiple parallel machine implementations.
- Implementation details of parallel machine confined to Comm classes.
- In particular, rest of Epetra has no dependence on MPI.

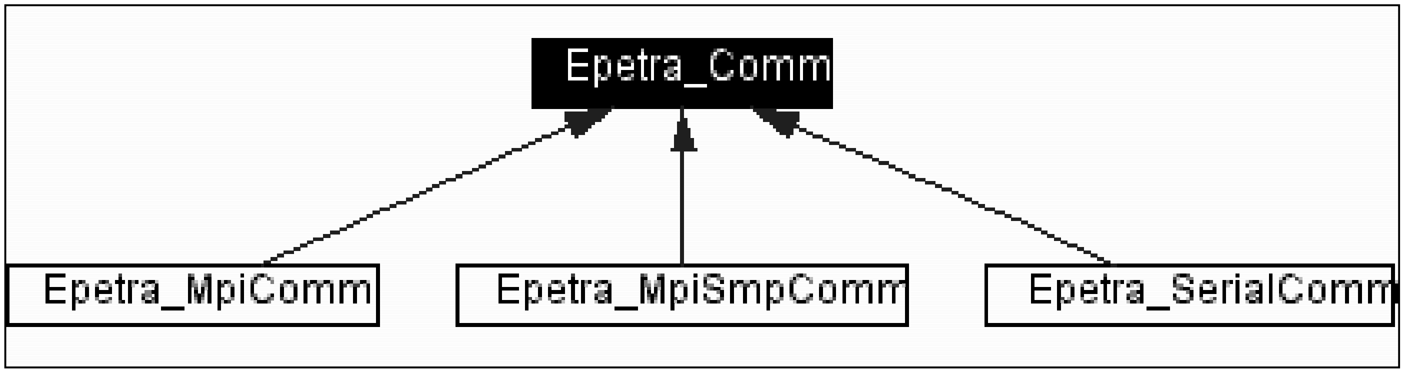
Sandia National Laboratories

Comm Methods

- **Barrier**() const=0 [pure virtual]
- **Broadcast**(double *MyVals, int Count, int Root) const=0 [pure virtual]
- **Broadcast**(int *MyVals, int Count, int Root) const=0 [pure virtual]
- **CreateDistributor**() const=0 [pure virtual]
- **GatherAll**(double *MyVals, double *AllVals, int Count) const=0 [pure virtual]
- **GatherAll**(int *MyVals, int *AllVals, int Count) const=0 [pure virtual]
- **MaxAll**(double *PartialMaxs, double *GlobalMaxs, int Count) const=0 [pure virtual]
- **MaxAll**(int *PartialMaxs, int *GlobalMaxs, int Count) const=0 [pure virtual]
- **MinAll**(double *PartialMins, double *GlobalMins, int Count) const=0 [pure virtual]
- **MinAll**(int *PartialMins, int *GlobalMins, int Count) const=0 [pure virtual]
- **MyPID**() const=0 [pure virtual]
- **NumProc**() const=0 [pure virtual]
- **Print**(ostream &os) const=0 [pure virtual]
- **ScanSum**(double *MyVals, double *ScanSums, int Count) const=0 [pure virtual]
- **ScanSum**(int *MyVals, int *ScanSums, int Count) const=0 [pure virtual]
- **SumAll**(double *PartialSums, double *GlobalSums, int Count) const=0 [pure virtual]
- **SumAll**(int *PartialSums, int *GlobalSums, int Count) const=0 [pure virtual]
- **~Epetra_Comm**() [inline, virtual]

Sandia National Laboratories

Comm Implementations



Three current implementations of Petra_Comm:

- Epetra_SerialComm:
  - Allows easy simultaneous support of serial and parallel version of user code.
- Epetra_MpiComm:
  - OO wrapping of C MPI interface.
- Epetra_MpiSmpComm:
  - Allows definition/use of shared memory multiprocessor nodes.
- PVM version in the future.

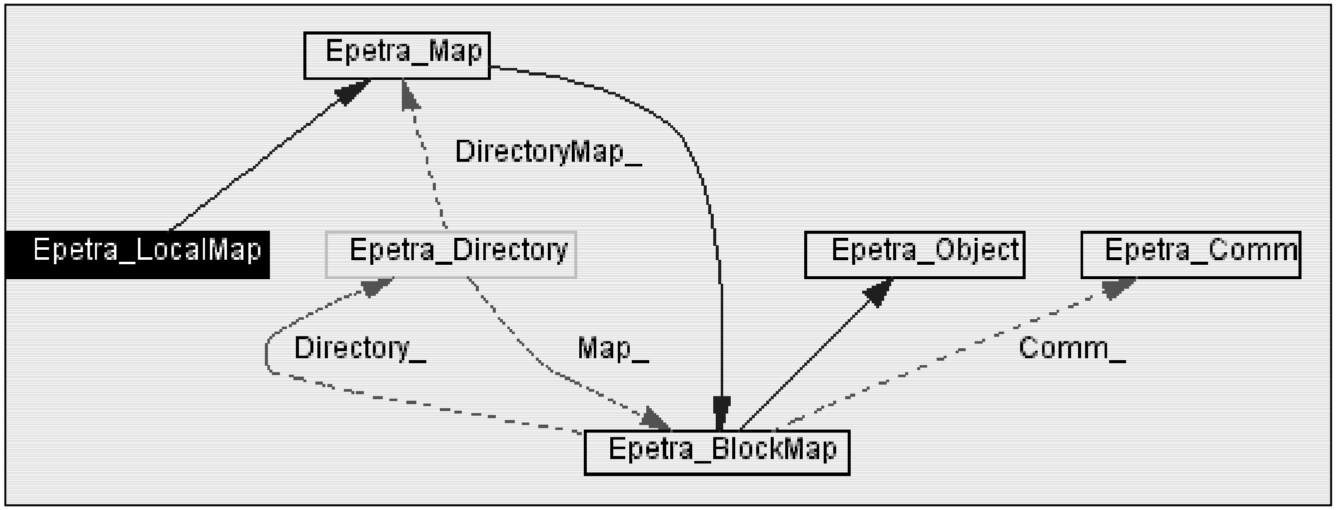
Map Classes

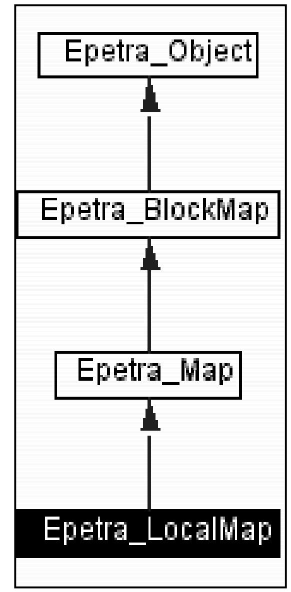
- Epetra maps prescribe the layout of distributed objects across the parallel machine.
- Typical map: 99 elements, 4 MPI processes could look like:
  - Number of elements = 25 on PE 0 through 2,
    = 24 on PE 3.
  - GlobalElementList = {0, 1, 2, …, 24} on PE 0,
    = {25, 26, …, 49} on PE 1. … etc.
- Funky Map: 10 elements, 3 MPI processes could look like:
  - Number of elements = 6 on PE 0,
    = 4 on PE 1,
    = 0 on PE 2.
  - GlobalElementList = {22, 3, 5, 2, 99, 54} on PE 0,
    = { 5, 10, 12, 24} on PE 1,
    = {} on PE 2.

Note: Global elements IDs (GIDs) are only labels:

- Need not be contiguous range on a processor.
- Need not be uniquely assigned to processors.
- Funky map is not unreasonable, given auto-generated meshes, etc.
- Use of a "Directory" facilitates arbitrary GID support.

Epetra Map Collaboration Diagram & Inheritance Graph





Notes:

1. **Epetra_Object is base class for all concrete Epetra classes:**
   - **Has labeling and ostream methods.**
   - **Maintains definitions of global constants.**
2. **BlockMap is the base map class.**
3. **Maps have Epetra_Directory to keep track of global ID distribution.**

Sandia National Laboratories

Types of Epetra Maps

- Two basic characteristic attributes:
  - **Local or not:**
    - A local map creates and maintains *replicated local objects:*
      - Object is the same across all processors.
      - Useful for some algorithms, Hessenberg matrix in GMRES, block dot products, etc.
    - Non-local creates *distributed global objects:*
      - Object is distributed across all processors. This is what we think of as a "standard" map.
  - **Block or not:**
    - Block supports variable weight per element.
    - Primarily used for sparse matrix whose entries are dense matrices.

BlockMap Ctors and Dtors

➢**Epetra_BlockMap** (int NumGlobalElements, int ElementSize, int IndexBase,
const Epetra_Comm &Comm)
Constructor for a Epetra-defined uniform linear distribution of constant block size elements.

➢**Epetra_BlockMap** (int NumGlobalElements, int NumMyElements, int ElementSize, int IndexBase,
const Epetra_Comm &Comm)
Constructor for a user-defined linear distribution of constant block size elements.

➢**Epetra_BlockMap** (int NumGlobalElements, int NumMyElements, int *MyGlobalElements,
int ElementSize, int IndexBase, const Epetra_Comm &Comm)
Constructor for a user-defined arbitrary distribution of constant block size elements.

➢**Epetra_BlockMap** (int NumGlobalElements, int NumMyElements, int *MyGlobalElements,
int *ElementSizeList, int IndexBase, const Epetra_Comm &Comm)
Constructor for a user-defined arbitrary distribution of variable block size elements.

➢**Epetra_BlockMap** (const Epetra_BlockMap &map)
Copy constructor.

➢virtual **~Epetra_BlockMap** (void)
Destructor.

Sandia
National
Laboratories

Some Map Methods

Local/Global ID accessor functions

int **RemoteIDList** (int NumIDs, const int *GIDList, int *PIDList, int *LIDList) const
Returns the processor IDs and corresponding local index value for a given list of global indices.

int **LID** (int GID) const *Returns local ID of global ID, return -1 if not on this processor.*

int **GID** (int LID) const *Returns global ID of local ID, return IndexBase-1 if GID not on this proc.*

Size and dimension accessor functions

int **NumGlobalElements** () const *Number of elements across all processors.*

int **NumMyElements** () const *Number of elements on the calling processor.*

int **MyGlobalElements** (int *MyGlobalElementList) const
Puts list of global elements on this processor into the user-provided array.

int **IndexBase** () const *Index base for this map.*

Sandia National Laboratories

Epetra Vector Class

- Supports construction and manipulation of vectors.
  - Distributed global vectors.
  - Replicated local vectors.
- Can perform common vector operations:
  - Dot products, vector scalings and norms.
  - Fill with random values.
- Used with the Epetra Matrix classes for matrix-vector multiplication.
- Use in a parallel or serial environment is mostly transparent.
- Specialization of the Epetra MultiVector class.

Epetra MultiVector Class

- A multivector is a collection of one or more vectors with the same memory layout (map).
- Useful for block algorithms, multiple RHS, replicated local computations.
- A generalization of a 2D array:
  - If the memory stride between vectors is constant, then multivector is equivalent to 2D Fortran array.
  - Can wrap calls to BLAS, LAPACK in this class.
- Provides most of the implementation for the Epetra Vector class.

Epetra Vector/MultiVector Inheritance Graph

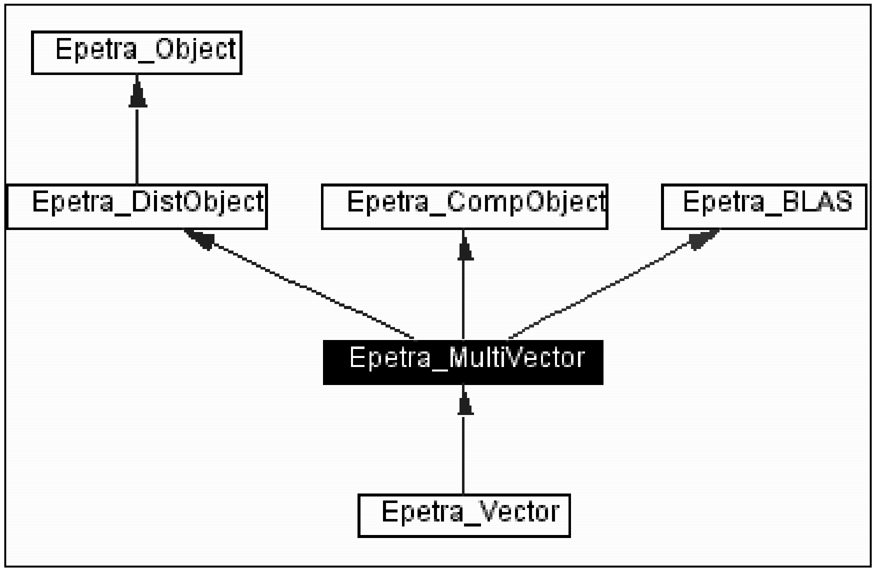
Notes:

1. **Vector is a specialization of MultiVector.**
   - **A multivector with one vector.**
2. **MultiVector *isa*:**
   a) **Distributed Object.**
      - **Data spread (or replicated) across processors.**

   b) **Computational Object.**
      - **Floating point operations occur (and will be recorded if user desires).**

   c) **BLAS Object.**
      - **Uses BLAS kernels for fast computations.**

   d) **More on common base classes later…**



Sandia National Laboratories

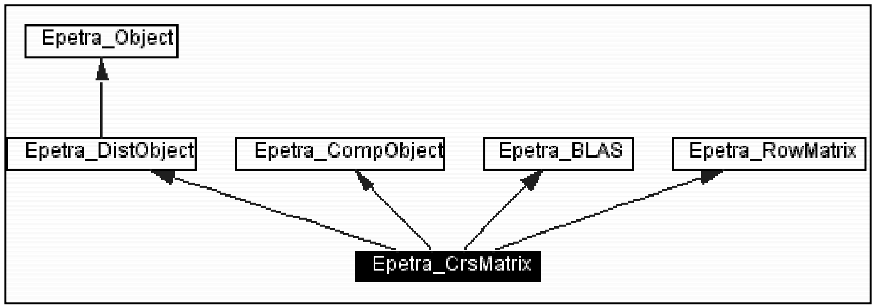
Epetra CrsGraph Class

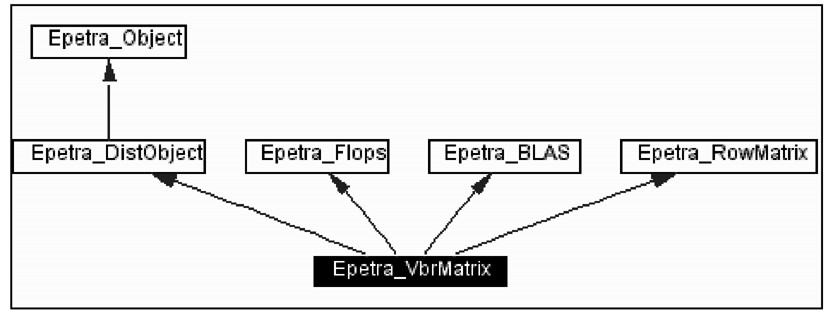
- Provides “skeletal” information for both sparse matrix classes (CRS and VBR).
- Allows *a priori* construction of skeleton that can be used by multiple matrices and reused in future.
- Provides graph information used by some load balancing tools.
- Exists in one of two states:
  - Global index space.
  - Local index space.

Epetra Matrix Classes

- Support construction and manipulation of:
  - Row based (Epetra_CrsMatrix) and
  - Block row based (Epetra_VbrMatrix) matrices.
- Constructors allow:
  - row-by-row or entry-by-entry construction.
  - Injection, replacement or sum-into entry capabilities.
- Supports common matrix operations:
  - Scaling.
  - Norms.
  - Matrix-vector multiplication.
  - Matrix-multivector multiplication.

Matrix Class Inheritance Details





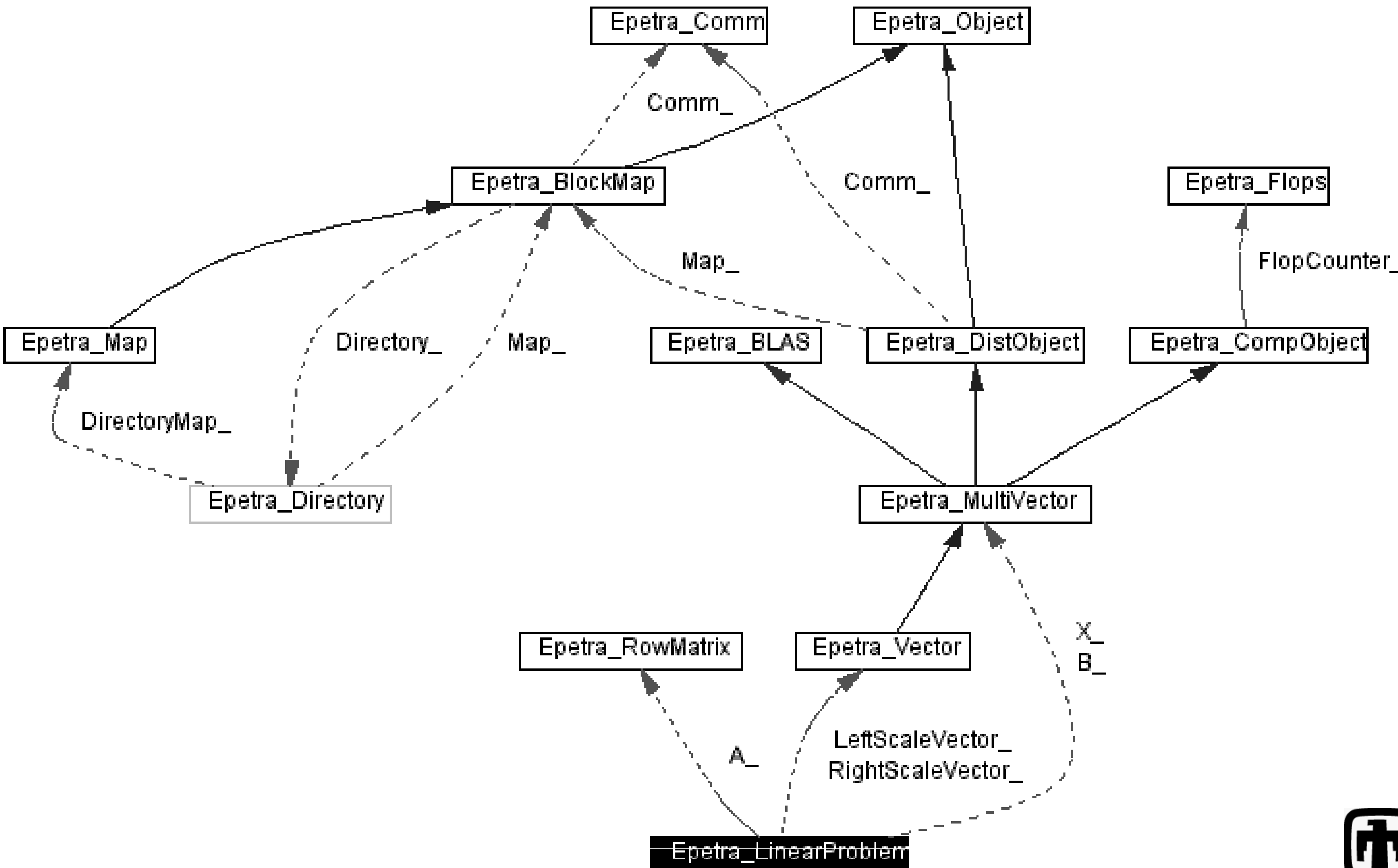
CrsMatrix and VbrMatrix inherit from:

- **Distributed Object: How data is spread across machine.**
- **Computational Object: Performs FLOPS.**
- **BLAS: Use BLAS kernels.**
- **RowMatrix: An object from either class has a common row access interface (used by AztecOO).**

LinearProblem Class

- A linear problem is defined by:
  - Matrix *A* : An Epetra_RowMatrix object. (but really a CrsMatrix or VbrMatrix object.)
  - Vectors *x, b* : Vector objects.
- To call AztecOO, define a LinearProblem:
  - Constructed from *A, x* and *b.*
  - Once defined, can:
    - Scale the problem (explicit preconditioning).
    - Precondition it (implicitly).
    - Change *x* and *b.*

LinearProblem Collaboration Diagram



Some LinearProblem Methods

Epetra_LinearProblem (**Epetra_RowMatrix** *A, **Epetra_MultiVector** *X, **Epetra_MultiVector** *B)
Epetra_LinearProblem Constructor.

void **SetOperator** (**Epetra_RowMatrix** *A)
Set Operator A of linear problem AX = B.

void **SetLHS** (**Epetra_MultiVector** *X)
Set left-hand-side X of linear problem AX = B.

void **SetRHS** (**Epetra_MultiVector** *B)
Set right-hand-side B of linear problem AX = B.

int **CheckInput** () const
Check input parameters for size consistency.

int **LeftScale** (const **Epetra_Vector** &D)
Perform left scaling of a linear problem.

int **RightScale** (const **Epetra_Vector** &D)
Perform right scaling of a linear problem.

Sandia
National
Laboratories

AztecOO

- Aztec is the workhorse solver at Sandia:
  - Extracted from the MPSalsa reacting flow code.
  - Installed in dozens of Sandia apps.
  - 800+ external licenses.
- AztecOO leverages the investment in Aztec:
  - Uses Aztec iterative methods and preconditioners.
- AztecOO improves on Aztec by:
  - Using Epetra objects for defining matrix and RHS.
  - Providing more preconditioners/scalings.
  - Using C++ class design to enable more sophisticated use.
- AztecOO interfaces allows:
  - Continued use of Aztec for functionality.
  - Introduction of new solver capabilities outside of Aztec.

Some AztecOO Methods

AztecOO (const Epetra_LinearProblem &problem) *AztecOO Constructor.*

int **SetAztecDefaults** () *AztecOO function to restore default options/parameter settings.*

int **SetAztecOption** (int option, int value) *AztecOO option setting function.*

int **SetAztecParam** (int param, double value) *AztecOO param setting function.*

int **Iterate** (int MaxIters, double Tolerance) *AztecOO iteration function.*

int **NumIters** () const *Returns the total number of iterations performed on this problem.*

double **TrueResidual** () const *Returns the true unscaled residual for this problem.*

double **ScaledResidual** () const *Returns the true scaled residual for this problem.*

A Simple Epetra/AztecOO Problem

```
// Header files omitted…
int main(int argc, char *argv[]) {
  MPI_Init(&argc,&argv); // Initialize MPI
  Epetra_MpiComm Comm( MPI_COMM_WORLD );

// ***** Map puts same number of equations on each pe *****

  int NumMyElements = 1000 ;
  Epetra_Map Map(-1, NumMyElements, 0, Comm);
  int NumGlobalElements = Map.NumGlobalElements();

// ***** Create an Epetra_Matrix  tridiag(-1,2,-1) *****

  Epetra_CrsMatrix A(Copy, Map, 3);
  double negOne = -1.0; double posTwo = 2.0;

  for (int i=0; i<NumMyElements; i++) {
    int GlobalRow = A.GRID(i);
    int RowLess1 = GlobalRow - 1;
    int RowPlus1 = GlobalRow + 1;
    if (RowLess1!=-1)
       A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowLess1);
    if (RowPlus1!=NumGlobalElements)
       A.InsertGlobalValues(GlobalRow, 1, &negOne, &RowPlus1);
    A.InsertGlobalValues(GlobalRow, 1, &posTwo, &GlobalRow);
  }
A.TransformToLocal(); // Transform from GIDs to LIDs
```

```
  // ***** Create x and b vectors *****
  Epetra_Vector x(Map);
  Epetra_Vector b(Map);
  b.Random(); // Fill RHS with random #s

// ***** Create Linear Problem *****
  Epetra_LinearProblem problem(&A, &x, &b);

  // ***** Create/define AztecOO instance, solve *****
  AztecOO solver(problem);
  solver.SetAztecOption(AZ_precond, AZ_Jacobi);
  solver.Iterate(1000, 1.0E-8);

// ***** Report results, finish ***********************
  cout << "Solver performed " << solver.NumIters()
        << " iterations." << endl
        << "Norm of true residual = "
        << solver.TrueResidual()
        << endl;

  MPI_Finalize() ;
  return 0;
}
```

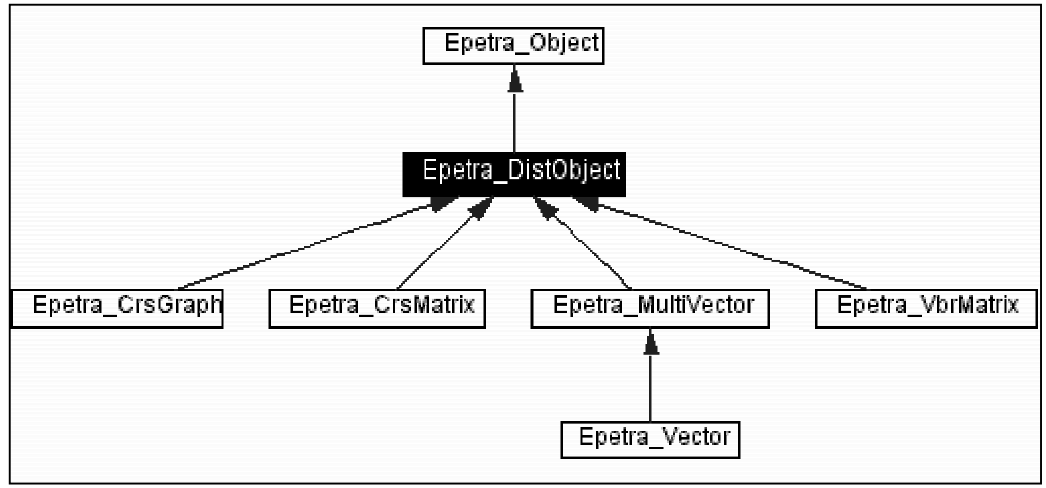
Sandia National Laboratories

Additional Epetra Classes: Utility and Base

- This completes the description of the basic user-oriented Epetra classes.
- Next we discuss some of the base and utility classes.

Epetra DistObject Base Class

- Epetra has 5 user-oriented distributed object classes:
  - Vector
  - MultiVector
  - CrsGraph
  - CrsMatrix
  - VbrMatrix



- DistObject is a base class for all the above:
  - Construction of DistObject requires a Map (or BlockMap or LocalMap).
  - Has concrete methods for parallel data redistribution of an object.
  - Has virtual Pack/Unpack method that each derived class must implement.
- DistObject advantages:
  - Minimized redundant code.
  - Facilitates incorporation of other distributed objects in future.

Epetra_DistObject Import/Export Methods

int **Import** (const Epetra_DistObject &A, const **Epetra_Import** &Importer, **Epetra_CombineMode** CombineMode)
*Imports an Epetra_DistObject using the **Epetra_Import** object.*

int **Import** (const Epetra_DistObject &A, const **Epetra_Export** &Exporter, **Epetra_CombineMode** CombineMode)
*Imports an Epetra_DistObject using the **Epetra_Export** object.*

int **Export** (const Epetra_DistObject &A, const **Epetra_Import** &Importer, **Epetra_CombineMode** CombineMode)
*Exports an Epetra_DistObject using the **Epetra_Import** object.*

int **Export** (const Epetra_DistObject &A, const **Epetra_Export** &Exporter, **Epetra_CombineMode** CombineMode)
*Exports an Epetra_DistObject using the **Epetra_Export** object.*

Epetra_DistObject Virtual Methods

virtual int **CheckSizes** (const Epetra_DistObject &Source)=0
Allows the source and target (this) objects to be compared for compatibility, return nonzero if not.

virtual int **CopyAndPermute** (const Epetra_DistObject &Source, int NumSameIDs,
int NumPermuteIDs, int *PermuteToLIDs, int *PermuteFromLIDs)=0
Perform ID copies and permutations that are on processor.

virtual int **PackAndPrepare** (const Epetra_DistObject &Source, int NumExportIDs, int *ExportLIDs,
int Nsend, int Nrecv, int &LenExports, char *&Exports, int &LenImports,
char *&Imports, int &SizeOfPacket, **Epetra_Distributor** &Distor)=0
*Perform any packing or preparation required for call to **DoTransfer**().*

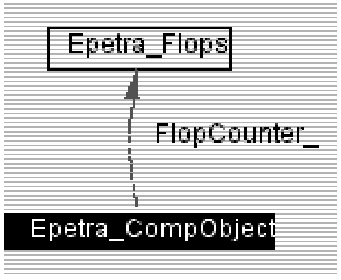
virtual int **UnpackAndCombine** (const Epetra_DistObject &Source, int NumImportIDs, int *ImportLIDs,
char *Imports, int &SizeOfPacket, **Epetra_Distributor** &Distor,
Epetra_CombineMode CombineMode)=0
*Perform any unpacking and combining after call to **DoTransfer**().*

Epetra_Time and Epetra_Flops

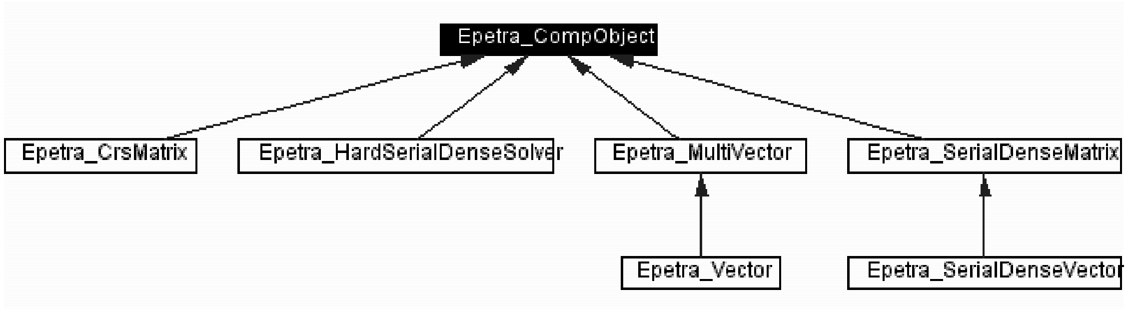
- **All Epetra computational classes count floating point operations (FLOPS):**
  - **FLOPS are associated with the *this* object.**
  - **Op counts are serial counts, that is, independent of number of processors.**
  - **Each computational class have a FLOPS() method that can be queried for the flop count of an object:**

```
Epetra_Vector V(map);
Epetra_Flops counter;
V.SetFlopCounter(counter);
V.Random();
V.Norm2();
double v_flops = V.Flops(); // v_flops should = len of V
```

Epetra_CompObject Class



- Epetra has 8 user-oriented distributed object classes:
  - Vector
  - MultiVector
  - CrsMatrix
  - VbrMatrix
  - SerialDenseVector
  - SerialDenseMatrix
  - SimpleSerialDenseSolver, HardSerialDenseSolver



- CompObject is a base class for all the above:
  - Trivial constructor.
  - Manages pointer to an Epetra_Flops counter object.
  - Allows a computational object to donate its FLOPS to a specified counter.
  - Any number of objects can be associated with a single counter object.

Epetra Serial Dense Matrix and Vector Classes

Epetra provides two types of serial dense classes:

- **(Thin)**
  - **Epetra_BLAS, Epetra_LAPACK:**
    - **Provide thin wrappers to BLAS and LAPACK routines.**
    - **A single interface to any BLAS routine (There is one call to DGEMM in all of Epetra).**
    - **A single method for all precision types. (GEMM covers SGEMM, DGEMM, CGEMM, ZGEMM) Helps with templates.**
    - **Inheritable: Any class can be a BLAS, LAPACK class.**
- **(OO)**
  - **Epetra_SerialDenseMatrix, Epetra_SerialDenseVector:**
    - **Fairly light-weight OO Dense matrix and vector classes.**
  - **Epetra_SimpleSerialDenseSolver:**
    - **Straight-forward LU solver.**
  - **Epetra_HardSerialDenseSolver:**
    - **Careful implementation that provide OO access to robust scaling and factorization techniques in LAPACK.**
  - **SPD versions of above.**

Sandia National Laboratories

Parallel Data Redistribution

- Petra vectors, multivectors, graphs and matrices are distributed via one of the map objects.
- A map is basically a partitioning of a list of global IDs:
  - IDs are simply labels, no need to use contiguous values (Directory class handles details for general ID lists).
  - No *a priori* restriction on replicated IDs.
- Given:
  - A source map.
  - A set of vectors, multivectors, graphs and matrices (or other packable objects) based on source map.
- Redistribution is performed by:
  - Specifying a target map with a new distribution of the global IDs.
  - Creating Import or Export object using the source and target maps.
  - Creating vectors, multivectors, graphs and matrices that are redistributed (to target map layout) using the Import/Export object.

Sandia National Laboratories

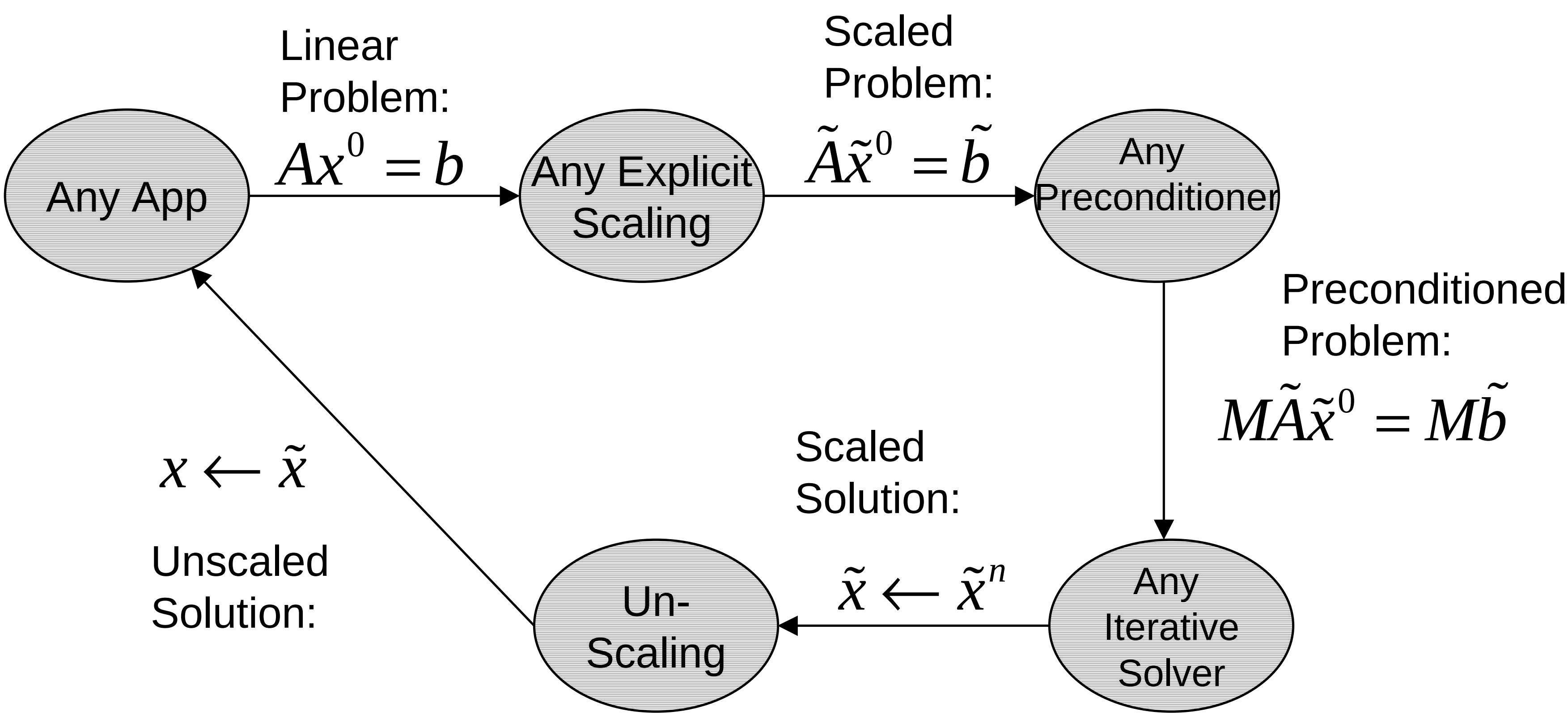
Import vs. Export

- Import (Export) means calling processor knows what it wants to receive (send).
- Distinction between Import/Export is important to user, almost identical in implementation.
- Import (Export) objects can be used to do an Export (Import) as a reverse operation.
- When mapping is bijective (1-to-1 and onto), either Import or Export is appropriate.

Sports Interview Component Model

Sandia National Laboratories

Linear Solver Component Model



The Trilinos Solver Framework (TSF)

- TSF specifies a set of abstract classes for:
  - ParameterList (simple database).
  - Multivector (generalization of vector).
  - Operator. (base transformation class).
  - Problem (primary control class).

  - And specializations of these classes.

- These interfaces prescribe:
  - What these objects should do.
  - How they are related.

- *But do not specify the implementation.*

TSF Abstract Class Hierarchy

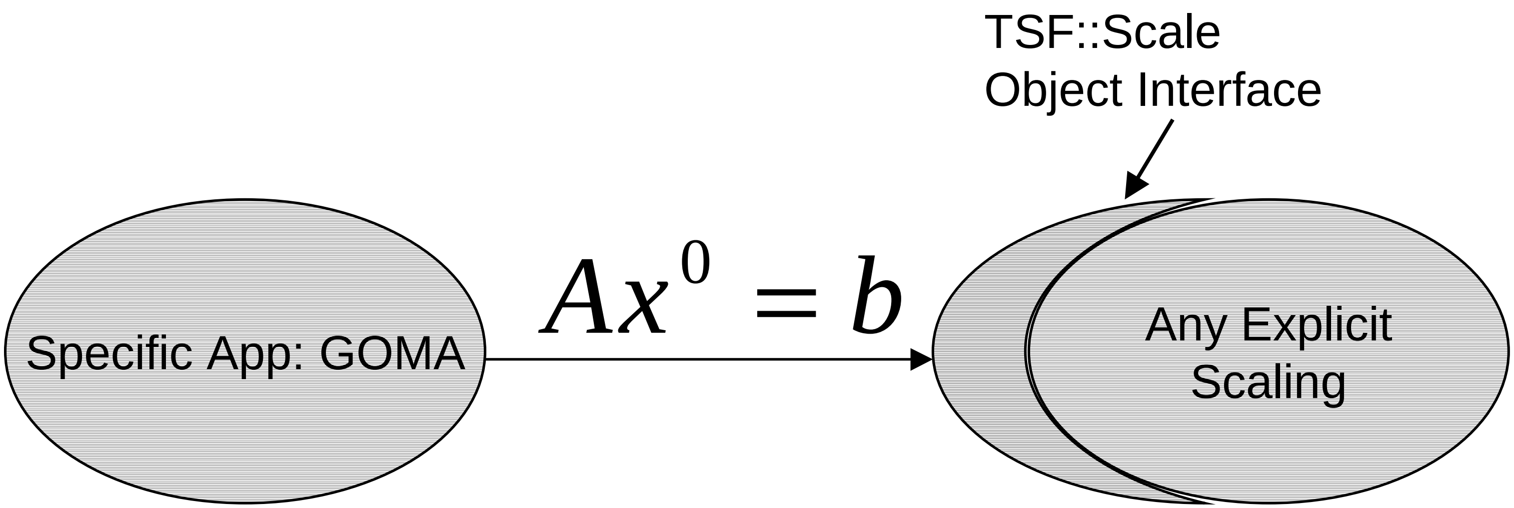
- TSF::ParameterList – Encapsulates parameter information for solvers.

- TSF::MultiVector – Abstract multivector class.
  - TSF::Vector – Specialization of MultiVector.

- TSF::Operator – Most basic of transformation classes.
  - TSF::LinOperator – Specialization of Operator.
    - TSF::Matrix – Specialization of LinOperator.
      - TSF::RowAccessMatrix – Specialization of Matrix.
  - TSF::Preconditioner
    - TSF::Scale
  - TSF::Solver
    - TSF::LinSolver
      - TSF::IterLinSolver

- TSF::Problem – Encapsulates all required info to define problem.
  - TSF::LinProblem
    - TSF::PrecLinProblem
  - TSF::EigenProblem
  - TSF::NonLinProblem

Abstract Interfaces



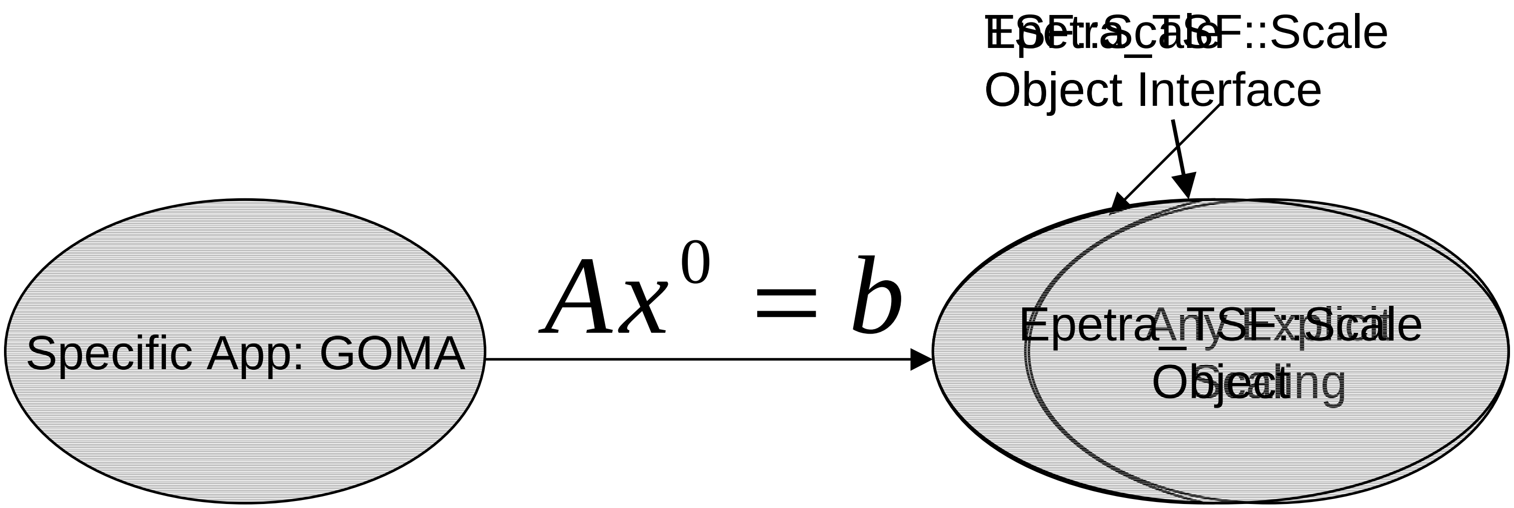
- **GOMA can use TSF::Scale to define scaling without specifying implementation.**
- **However, we need real code to make this work…**

Sandia National Laboratories

Concrete and Adaptor Classes

- **Essential Epetra (Epetra) is a library of *concrete* C++ classes.**
- **Several Epetra classes provide a variety of scaling algorithms:**
  - ➢ **Real, working code.**
  - ➢ **Parallel, distributed memory.**
  - ➢ **Numerically robust.**
- **To use Epetra with TSF::Scale:**
  - **Write an *adaptor* class: Epetra_TSF::Scale.**
    - **Note: An Epetra_TSF::Scale object**
      - ***isa* TSF::Scale object (implements TSF::Scale interface).**
      - ***isa* Epetra object (calls Epetra methods).**
- **Note: PETSc, LAPACK, others also provide scaling (equilibration) techniques.**

Epetra_TSF Adaptor Class



- **The Epetra_TSF adaptor class:**
  - **Uses Epetra for functionality.**
  - **Satisfies the interface needs of Goma.**
  - **Requires only a relinking of Goma (or a change in the *Factory* options).**
- **Using LAPACK or PETSc would require a similar (small) amount of work to integrate into TSF.**

Sandia National Laboratories

Trilinos Summary

Trilinos provides a flexible model for delivering solver capabilities:

- TSF Abstract Classes:
  - Provides a single interface for applications.
  - Gives apps access to any solver implementing TSF.
  - Allows algorithm developers to use generic programming.
- Concrete component class:
  - Epetra, AztecOO, Anasazi, IFPACK, ML, etc.
- Adaptor classes:
  - We always have a default TSF implementation using our concrete classes (Epetra, etc.).
  - Can easily integrate new solver components with minimal code development.

Trilinos and the Outside World

- **ESI (Equation Solver Interface):**
  de facto standard solver interface.
  - **Epetra and AztecOO provide the first ESI-compliant implementation (thanks to Alan Williams).**
- **TAO (Toolkit for Advanced Optimization):**
  Argonne optimization package.
  - **Epetra/AztecOO are being used (via ESI interface) for TAO solver services, along with PETSc implementation of ESI.**
- **CCA (Common Component Architecture):**
  Community effort to develop scientific SW components.
  - **Epetra/AztecOO to become a CCA solver component.**
- **Public Release of Trilinos/Epetra:**
  - **We will release Trilinos/Epetra this fall/winter.**
  - **Using LGPL for licensing.**
  - **ML, IFPACK , AztecOO, Komplex, Anasazi , NLS will be (or are) going through the same release process.**